Bryan Call:

All right, well I'm excited to be with you here this afternoon. As was said, my name is Bryan Call. I'm a Senior Principal Engineer with Route and I'm here to talk to you about Route's story and journey with CockroachDB. I want to start off by telling you a tale of two storage failures. The first one is a tragic tale of massive hard drive failure and the downtime and data loss and ruined weekend that that brought on from a traditional SQL database. The other story we'll talk about involves CockroachDB and to be honest, it's pretty boring because with Cockroach when you have a massive hardware failure, there's not a lot of drama, but it's a great comparison. So this first story happened to me a few years ago. It started on a Saturday afternoon. I was on my couch enjoying my weekend, hoping to just have a chill weekend when I get a text from my data center.

I was working at a small startup. There were only a few engineers, but we had a fairly large footprint in this data center and they texted me to say, "Hey, there's been a fire suppression event at the data center but everything's okay." And I am going, "I think this is a sign my weekend's going up in smoke." And it was followed up by a phone call from the SAN vendor who said, "Hey, we just got a phone home from your SAN that said there was a controller failover. Do you know what's going on?" And I said, "Not yet, but I think I'm about to find out." And come to find out, a technician had accidentally discharged the Inergen fire suppression system in this data center and Inergen is supposed to be safe around computer equipment, but what they didn't realize is the high pressure release would create a harmonic resonance wave that would damage hard drives throughout the data center.

And so, I had one of those scenarios where you have this array of 20 hard drives and you go, "When am I ever going to lose 11 hard drives all at once?" And well, yeah, I found out how that can happen. This was a typical SQL database and all I could do is replace the server and restore from backup and it was a ruined weekend and not a lot of fun, especially because back then database servers, we didn't have this cattle not pets thing. It was a pet and it was a pain to replace. We'll contrast that to an event that happened, oops, I went too far. Can we go backwards here? All right, an event that happened a couple of weeks ago, it was a Thursday afternoon and I logged in to just check on our Cockroach cluster, see was going on and there was this really weird pattern and it wasn't a big problem, but it was abnormal and a little bit of investigation and I wasn't really sure what was going on.

So I opened up the support case to see if a support engineer could help me figure out what this new pattern meant. And I got assigned to Daniel, give a shout-out to Daniel and support. I knew that I'd get a very thorough response from him because we've had some great interactions in the past. And so, I went back to my afternoon waiting for his response. He came back and said, "Hey, it looks like you're having a hardware failure. Maybe one of your drives is failing. You might try rebooting the server and if not, you may need to replace the node." I think, "Well, this is easy. I'm on AWS, I have Cockroach. I'll just replace the EC2 instance. That's quick, easy, no big deal." Didn't solve the issue. So I said, "Okay, well, that's fine. I can take the node offline. I can delete the EBS volume, reapply my terraform and it'll be back up and running." Which is exactly what happened.

I spent the evening watching the ranges re-replicate from the failed drive, which is boring, but let me work on the slide deck and we were back up. Nobody knew any better. No incident at work, nobody big deal because that's what happens when you have Cockroach. So why did I tell these stories? I just think this comparison of how we've progressed with database technology is pretty cool. But I jumped right into it, I didn't introduce myself. So a little bit about who I am. I am your stereotypical geek. I showing here a picture from high school with my first SQL server box. I'm sure we all took photos of our first SQL server box back when we were in high school. I know I did. It all started for me in a dot com basement startup. You got to have a good dot com basement startup story. I first started with a Postgres 95 actually and Cybase and then moved on to MS SQL, MySQL, DynamoDB and a whole slew of other database technologies.

I love code and I love infrastructure. I've always preferred to be full stack as I call it, which doesn't mean I know CSS and HTML. It means I understand everything from the hardware where my code is running all the way up down to the code and the business value that it provides. And I've, my home has been Route for the last four years. One of my favorite places I've worked, had an interesting journey there. I started off in leadership and led teams that were responsible for CockroachDB and then got a little bit antsy, wanted to get back hands-on technically, took a role change and now I'm the engineer principally responsible for our main Cockroach cluster. So let's talk a little bit about Route. You may not have heard of Route. Route is the leading post-purchase customer experience solution, protecting brands and their customers with shipping insurance, package tracking, fast issue resolution, carbon-neutral shipping and remarketing.

That's right from our marketing deck and it talks about what we do, but maybe not why we do it. And I stole this slide as well because it tells a story that I like, which is, Route is all about connecting brands with their customers in meaningful ways and helping those brands provide truly great shopping experiences and stories for their customers. We've been around about five years. We serve about 13,000 merchants and we're all about deeply connecting brands with their customers. Now to do that, means we've got to have a lot of data. And so, this gets into the meat of this talk, which is how CockroachDB has helped us deliver on Route's mission. Since I joined the company, we always talk about everything you order in one place. Route's all about giving consumers everything they order online regardless of who they bought it from, keeping track of that, keeping track of those packages, when are they going to arrive, where are they at, all in one place, in one convenient app.

And fulfilling our mission of everything you order in one place means that well, all the data needs to be in one place, one place that can handle it all. So our data platform has to scale. Next part of our mission is, like every company, we want to help our customers increase their revenue. We want to help the merchants and brands we partner with be able to increase their revenue, but that only happens if they're up and we're up. And so, our data platform absolutely has to be always on. And while we have lots of different things we do, it's not just about the data platform being resilient to keep our customers up, it's absolutely a key part of the equation. And the last thing I want to talk about today is increased customer satisfaction. Route is all about delivering positive customer experiences for merchants when the worst happens.

When something goes wrong with a shipment, we're there to make it right. And spinning load icons, well, they're not a positive experience, so our data platform has to perform. So we'll talk about each of these in a little bit more depth. Starting off with everything you order in one place. From the beginning we knew we had these big aspirations. We're a small company, big aspirations. We knew we'd need to deal with large amounts of data and our use case meant that data would be relational in nature. We knew that having everything in one place physically and logically in a single database sure would make things easier. Sure, I can present everything as if it's in one place even though it's not but that's a lot more work and complexity and things that can go wrong. So if you can find a data platform that lets you unify it automatically, that sure is nice.

The dual merchant and consumer focus at Route is one of the things I love the most but it introduces some interesting challenges because at other SaaS companies I've been at, we could shard data in different ways, but if I try and shard data by merchants, it makes it harder to provide a holistic consumer view. If I try and shard by consumers, it makes it harder to provide a holistic merchant view. And so, a lot of those traditional techniques would be, not work for us. Our CTO at the time I joined, Ryan Debenham wanted a database that would handle scaling and sharding for us without requiring we build it ourselves.

He had done that at prior places and so the decision to use Cockroach at Route actually predates me. It was something I walked into. We're a few months into having CockroachDB and then it was dropped on my plate.

And as I think about it, I wanted to make, compare this journey that Route had with CockroachDB to my journey as a runner because they actually correlated about the same timeframe. I decided I was going to run my first 5K. This is a picture of the 5K course. This is actually on Disney's private island in The Bahamas. That's what got me motivated to run was this idea of running on a private island, The Bahamas, that sounds really cool. It's not cool. It's hot, it's humid, it's a miserable place to run, but I still do it for nostalgia stake. But I did that run and I never thought in a million years I'd try to run a farther distance than that. But when we started out, Ryan knew that we needed a marathon grade database. We were maybe a 5K company, but we needed a marathon grade database and so, he pushed for us to implement that early on.

And much like training for a long distance race, our Cockroach journey has involved some sore muscles and some exhausting long runs along the way. But just as I discovered, this is a picture from my second half-marathon, which actually ran with Ryan, it's all worth it. When you put in that training, it becomes so worth it. And so, 14 half-marathons later, I truly understand the term runner's high and four years later and I truly understand why choosing CockroachDB was worth the effort. So let's talk a little bit about how we've scaled. So Route's been the fastest growing startup that I've had the privilege to work for and this is some numbers around our scale. We've 3x-ed our node count over the time I've been here, 12x-ed our CPU count. We've got 52 terabytes of storage now available.

And all of this scaling and growth has involved no painful data migrations and no required downtime. Now I've never worked with a database before where I was able to add storage, add CPU, do all these things and no data migrations, no downtime, it all stayed online while I did it, it's pretty incredible and that's why I'm here to talk to you today. And at leads into my second point, which is about increasing revenue because in the e-commerce space where we work, you can't increase revenue if you take things down. If users can't check out, you're not making money. And while we have a lot of different layers of redundancy and things we do to make sure that we stay up and never interfere with that, having a database that stays up is definitely a key part of that. And so, I think about CockroachDB in our journey., we've never had a major Black Friday or Cyber Monday outage.

I remember the first time I realized the power of this. I had challenged a team, a particular engineer to really learn Cockroach and understand it coming into our busiest time of the year, which was this Black Friday, Cyber Monday season and that day I had stayed at home. I was online watching, making sure everything was going well and things had been going well and I knew the crowds had died down. I needed something at Walmart. I'm like, I'm going to go risk going out to Walmart, it's going to be okay. And I pull in the parking lot and I get a text, "One of our Cockroach nodes just went down." And I go, "Oh no." And, "I am going to have to go back home. This is bad. This is bad. I shouldn't have gone. I jinxed the team."

And I get a text, "Oh, it's back up, Jordan got it back up. Everything's good. Nothing to worry about." Nobody knew, nobody noticed. That's when I knew that we had made a good choice and that we were on the right path. We survived multiple hardware type failure issues in AWS. Just because in the cloud and you don't have to think about the hardware, doesn't mean it doesn't exist and doesn't mean it doesn't fail. And that's not a knock on AWS, it's just reality. We've performed numerous scaling operations, hardware migrations, and even a complete infrastructure re-architecture without downtime. As I heard Spencer talking about in his talk about multi-cloud this morning, it hit me because we went through this process. We didn't change clouds, but we completely replaced every Terraform resource we had and re-architected how we ran on AWS.

And we did that while running both the new and the old simultaneously with zero downtime and migrated over time and it was a non-event. No maintenance window, no downtime, nothing. We could have just easily spanned clouds or gone from a private data center or any number of things and that same process would've worked because that's the power that Cockroach gives you and I love that. It's been so cool. And maybe this is my favorite. Jack, I think, alluded to this in his talk a minute ago, that patch and upgrade regularly without traditional maintenance windows. Some of my fellow engineers when they have to do work on some of their databases, we do run different databases besides Cockroach for different workloads. They have to do a maintenance window. And I know in the past I've always been scared to touch the big database server because you're wondering what's going to go wrong? Is this going to turn into a horrible night? And now I patch Cockroach whenever I feel like it.

Maybe that comes across cavalier for an important system and then I'm not cavalier about it, but we have such confidence and we've done it over and over again and we know we can just do it when we need to do it and it's not a scary thing. I need to talk a little bit about how this happens. And so, the Cockroach experts in the room may say have simplified things a little bit too much, but I want to take a minute and talk a little bit about Cockroach's architecture and my view of it at least. And when I started learning about relational databases, they always use this example of a student information system. I guess they figured students understood that problem domain. So we're going to imagine we have a student table and Cockroach is going to shard that table. And so, one way to think about it is, it's going to put all the students with a last name of A through F in one range. It's going to put the G through in another range and the P through Z in another range.

Then it's going to replicate each of these ranges based on the replication factor. And this example replication factor of three means each range is replicated three times. And then each one of those replicas of a range has to be put onto a store. And for simplicity, replica one is on store one, replica two on store two and replica three on store three. And these stores could correspond, for example, to an EBS volume. What this allows us to do is, we can lose any one of those replicas and life goes on because we've got them somewhere else. We can lose any one of those EBS volumes and life goes on because we've got those replicas somewhere else. This is the underlying architecture that makes all this possible. Now, if I want to talk about how it works at Route, we'll talk a little bit about our architecture. Now, we are a self-hosted with Cockroach because when we started the cloud offering really wasn't around or hadn't really been developed yet when we started with CockroachDB.

And so, we're still running self-hosted. We're in AWS and we're an ECS shop. We're not a Kubernetes shop. I know there's the Kubernetes operator in some automation, but we won't get into that debate here. But we're in ECS shop and we needed to make some changes. This is something else I'm going to pick on Spencer again and hope I don't lose my favorable pricing, but he alluded in his talk about some of the cool things you can do with EBS, like being able to have this persistent storage even if your compute nodes changed. And this was something we weren't taking advantage of and our data set had gotten so big that when we replaced a node, the replication time was painful. So we started rethinking about this and as I thought about it, I thought, "Okay, well, what we have is we have a group of EBS volumes that we're going to tag together and I'm going to make this tagged volume group. These are the EBS volumes that need to go together on a node."

And I'm a big Star Trek fan, so I envisioned it as the Starship Enterprise, okay. So I've got this Starship that represents my storage layer. And then the other thing, and we can have a debate after about, is the enterprise the enterprise because of the ship or is it the enterprise because of Jean-Luc Picard? That's another debate for after. But we've got our crew and our crew is represented here as an auto-scaling group in AWS with an EC2 instance that runs a Cockroach container orchestrated by ECS, right.

So we've got our crew, we've got our ship now we've got to bring those together. In the world of Star Trek, I would just use a transporter and beam the crew aboard the new ship and life would go on. Well, for us, we need to build our own transporter and our transporter leverages EC2 user knit scripts, which runs some custom tooling we've developed that goes and finds a tagged volume group of available volumes, mounts it on, exposes it, makes it available to ECS so when that container starts up, that storage is already there and I don't have to replicate everything just because I replaced my EC2 instance.

To look at this another way, we can look at this little architecture diagram of how we look in AWS. We've got availability zones, we've got these tagged volume groups with EBS volumes, we've got EC2 instances. I guess I should call out here explicitly, we do use multiple stores per Cockroach node to spread out our IO. It's better, we found it's better than having one giant EBS volume. We've got these auto-scaling groups that are running EC2 that are running Cockroach containers. This is all orchestrated with ECS and we've got our application load balanced then load balances to these different nodes. The idea here is, I can step on any one of these boxes, any one of these roaches, if you will, and the cluster survives and things stay online.

So if I lose a EBS volume, no big deal, lose an EC2 instance, hey, our auto-scaling group will replace that. Lose a container, obviously, that's easy to just restart. Anything I lose here, it doesn't matter. I can lose a whole availability zone and I don't go down. The other thing that's not shown on here is, we bunch a utility lamb does to do things like rotate our client certs and automatically generate debug zips when support asks for them, things like that. So with this architecture, we really can automate and deal with almost any failure that comes our way. So let's talk about some of the key learnings we've had as we've learned to fully leverage Cockroach's always on capabilities. One of the first things is, servers are roaches and not pets. This comes from that DevOps philosophy of cattle, not pets, but make sure your servers don't require special treatment or easily replace.

This is something as we've gotten more and more in this mindset, which is very different from the traditional database mindset in my experience. It makes you more resilient and makes life easier. Automate as much failure recovery as possible. You want to make sure if something goes wrong, yes, Cockroach can handle things going wrong, but you want to get your resiliency back. You want to recover from those failures so that just in case something else is going to fail afterwards as quickly as possible. And so, we automate as much of our failure recovery as possible. Train for the pain. This is a big one. You want to make sure you regularly simulate failures so you have confidence in how things will go. And the way we do this is our standard maintenance procedures. It is in a controlled way, but they're exercising a lot of our redundancy and resiliency through the way in which we patch and do regular maintenance.

For example, when I need to patch the underlying OS on one of my EC2 instances, I'm building a new AMI. I'm terminating an instance. I'm replacing it in a controlled fashion, but it simulates the same thing that the auto-scaling group would do if it actually needed to replace a failed node due to a hardware failure. So I have confidence that that works. And so, regularly simulating those failures has been important to us to build confidence. And then the last thing is, don't ignore to pull muscle. I've learned this as a runner. You got to take care of that so you don't find yourself out of the running game longer than you intended. And with Cockroach, you want to sure you follow up on those. Anything that might be going wrong while it's a small problem before it becomes a big problem.

That's a general rule with almost any system but we found it especially true with Cockroach because if we catch it early, then it never becomes a big problem. This leads to third point I want to talk about, increased customer satisfaction. Route helps our merchants increase their customer satisfaction, and Cockroach helps us give our internal data clients a positive experience. One of the first things I wanted to talk about, and this is something that Jack mentioned in his talk as well, and that's change feeds. One of the things I think a lot of us have learned, especially if you're more in the transactional database environment, is you really don't want your data scientists and your data people running queries on your server. Really, really not fun when they try and run their analytics workload on your transactional server. And so, we want to be able to separate that and that becomes so easy with Cockroach because of change feeds.

I can change data capture all my data over to S3 and then let my data team go do whatever they want to do to get that into Snowflake or Databricks or whatever else they're going to use. So we've been using change feeds almost, that was one of the first projects I saw implemented when I came to Route was getting our change feed up and running so that we could do this and they work great. Another thing is SQL language, right. It's so great that with Cockroach we have the SQL language to provide all that flexibility. A lot of these no SQL databases are cool, but if you want to change your query patterns and do things differently, now you got to store your data differently and often you got to go through a migration and it's painful and hard, whereas having access to the full SQL language gives us all of this cool capability.

But it does come with a little bit of a cost, right? We got to understand our query plans to keep our performance high. We got to understand what's going on underneath the hood because you can write some really gnarly, not good SQL and sometimes you pay the consequences of that. So I want to take just a minute to compare databases here, and I'm going to pick on DynamoDB as a representation of NoSQL. It is actually my favorite NoSQL at the moment. And we do use it. It's not a worthless database by any means. There's workloads where it makes sense, but with DynamoDB, you are the optimizer, okay. You are the one who is determining how that query is going to run. You can query by hash key, you can query by hash and sort key. You can decide to do a scan and look at every single record, but there's no optimizer involved because you at the API level, tell Dynamo exactly what you want it to do and how to find the data.

Now this leads to consistent query performance, but there's a lot of flexibility you lose. You have to think about things in very specific ways. Compare that to a traditional SQL database, right. And I've got a flexible query language, but I've got to have this optimizer that figures out how to answer my question in an effective manner, and which leads to variable query performance, right. And then I also am usually limited to a single server to execute that query. I can only use the resources of a single server in my query. And now we'll compare that to CockroachDB, right. With Cockroach, we've got the flexible query language, but we can now leverage the power of multiple servers because we can truly scale out our workloads with Cockroach. But this does mean we have to have a sophisticated optimizer. And this is something that we have learned sometimes the hard way. Cockroach makes things look like magic.

It's not magic, it's just a lot of really good engineering. But with really good engineering comes complexity, and you've got to understand that. And so, it leads to this question of, how do I optimize the optimizer? How do I make sure that my queries are running well? And the best answer is, you just ask Glenn or Seung. Now I am picking on Glenn and Seung because they're the two enterprise architects that I've had the pleasure of working with the most. Glenn was my first experience to enterprise architect at CockroachDB and Seung is the current glutton for punishment for all of my questions. He's assigned to our account and he's awesome. And too bad he couldn't be here today but having access to an enterprise architect is an invaluable resource. Now I'm somebody who hates paying for enterprise support and paying for upgraded support because I usually don't feel like I get the value out of it.

Cockroach has been the exception to the rule for me and in my career, it's been the best vendor from a enterprise support perspective that I've worked with and we've absolutely gotten an ROI and it's been worth the money. So I have to call out and shout out that team. But ultimately, what are enterprise architects going to do? Well, they're going to help you master the tools. They're going to help teach you how to use the metrics console, how to use SQL insights, how do use SQL diagnostics to understand what's going on in your cluster and be able to optimize and work with it well. It's just great to have them as guides along the way. I also have to give a shout-out to the engineering teams that work on these tools because as a longtime Cockroach customer, I can say they've gotten significantly better in the last couple of releases.

The team's really done a great job of focusing on making that better for their customers, and I really appreciate that. I also have to pause for a minute and give an ode to covering indexes. Now, this is a concept that's not unique to Cockroach, but Cockroach definitely is a place where you want to understand and use these more than I have with other databases that I've worked with. And we'll go into a little bit more depth on what I mean by a covering index and how that works in just a second. I had another shout out here to master the stats that table stats are keys to making your optimizer work well. And this is another great topic to talk about with enterprise architects. So let's talk a little bit about covering indexes. So in a traditional index, I've got a table and I've got an index, and maybe this index lets me look up by name.

So I can look up John's name, and in that index I'll have his ID, his 123 ID, that's the primary key. And if I want to get John's phone number, I can use that index to find his record but then I got to go look up in the table by the primary key to actually grab his phone number. And in a distributed system, there's a potential that these two bits of data are actually stored on different servers, different nodes, which can make that lookup be a little bit expensive more so than in a traditional database. With a covering index, I actually say, "Hey, I want you to store the phone number right adjacent with the record in the index, even though I'm not going to search by it, I want you to have it right there so when I ask you to look up John's phone number, you can look in one place and give it to me."

So this concept exists in other SQL databases for sure, but the cost of that primary key lookup is potentially higher when you're dealing with a distributed database. And so, it becomes even more important to understand and leverage these indexes. And so, that's why I found them especially worthwhile when working with Cockroach. Now like any other tool, there's pros and cons. You got to understand it. Real quick, the syntax, if you're not familiar with doing this, is to add a storing clause under your end of your index. So you create this index on table people for name and storing phone number or whatever columns you want to store. That's how, that's the syntax for it. Now you are duplicating storage and IO for all these stored columns. So that means that anytime we update John's phone number, now I have to update it in the index node and in the primary table, which is an extra IO, right?

And that's not always something you want to do. And so, it's not great for highly volatile columns. It's not great to do on a whole ton of columns or if you're going to be writing more than reading, but it's a really valuable tool that we've found is really helpful, especially as your dataset grows is something to be aware of. Also, [inaudible 00:24:29] SQL Insights likes to recommend these indexes, but it tends to be a little bit aggressive in my experience in recommending them. And so, you got to take those recommendations with a grain of salt. You don't necessarily want to include every single column. And it can also recommend highly volatile columns where you know the cost of keeping them up to date is not going to be worth it. But it's definitely a helpful place to identify potential places to do that. All right, I've talked a lot, said a lot of positive things about CockroachDB, but it was mentioned at the very beginning of this about authenticity.

And I want to be an authentic customer and talk about the realities of our journey with Cockroach. This is a picture from the last half marathon I ran when I hit mile 10. For half-marathons when I hit mile 10, that's where I really started to hit a wall and really start to feel the pain and question my life choices. But I found as I push through that pain from mile 10 to the finish, that's when I can get the big rewarded. The reality is we've had a number of mile 10 moments with CockroachDB where we've had to work through issues. It's been hard. We've hit bugs or we've hit our own inexperience or mistakes we've made and then had to pay for them. But I have to say that the payoff's great. This is a picture from a race I ran at Disney World.

It's my favorite finish line picture ever and yes, I dressed like Donald Duck to run a race, but it was really, really fun. But I have to say that much like those moments where I've pushed through the pain and mile 10 and come to the finish line and been so glad I did the race, as we've pushed through the pain, I'm so glad that we've embraced CockroachDB. We've had to take time to learn the technology. We found that staying engaged with our account team is especially valuable. We've learned the importance of keeping things well-maintained. There's a period of time where we weren't paying as much attention to Cockroach as maybe we should have, and that can bite you. But by investing in our decision, we have totally reaped the rewards and seen the benefits of using CockroachDB. It's been a key part of our journey as a company and being able to scale and handle the load and really be an enterprise ready solution because we early on made the decision to embrace this new database, CockroachDB. And I wanted to end with a final story about repeating the benefits.

This is a picture of the Statue of Liberty I took. I know Cockroach has a big office in New York. We took my family to New York for the first time this last October, and we had a chance to take a cruise out of New York City and sail by the Statue of Liberty. And for me, Cockroach has given me personally, the liberty to not worry about our most critical database going down due to some random hardware failure. And in fact, while I was enjoying Bermuda's Pink Sand Beaches an EBS volume was failing. No one was trying to reach me. I had cell service and no one was anxious to tell me about it when I got back. It was a non-event for the team. And some of this is because I've worked with some of my favorite and most talented engineers in my career at Route.

We've got a really great team, and I knew they wanted me to have my vacation, but a lot of it was because the team was able to very quickly recover without downtime, without a big incident because we were running on CockroachDB. And this to me was just another example of how Cockroach provides always on data and the freedom to rest easily. And well, that's all folks.